A System to Provide Deterministic Flight Software Operation and Maximize Multicore Processing Performance: The Safe and Precise Landing – Integrated Capabilities Evolution (SPLICE) Datapath

David Rutishauser
Avionic Systems Division
NASA Johnson Space Center
Houston, U.S.A.
david.k.rutishauser@nasa.gov

John Prothro
Avenue Techologies and Commodities Inc.
Oklahoma, U.S.A.
john.r.prothro@nasa.gov

Jordan Fail
Avionic Systems Division
NASA Johnson Space Center
Houston, U.S.A.
jordan.w.fail@nasa.gov

***Abstract*— A method and design are described for a system that processes multiple data streams, utilizing a multicore asymmetric processing architecture, that eliminates data interrupts to the application processors. The design supports a deterministic environment for flight software in NASA's Safe and Precise Landing – Integrated Capabilities Evolution (SPLICE) project. The SPLICE project develops sensor, algorithm, and compute technologies for Precision Landing and Hazard Avoidance (PL&HA) capabilities. The compute technology for SPLICE is the Descent and Landing Computer (DLC). The DLC hosts several SPLICE algorithms with high computational resource requirements that must be executed in a real-time and deterministic manner. The software runs on a custom Single Board Computer (SBC), with a Xilinx Ultrascale+ Multiprocessor System-on-a-Chip (MPSoC). Input data for the flight software is from a variety of sensors, unique with respect to data rate and packet size. A data path between the SPLICE sensors and algorithms is designed to efficiently deliver this data to the flight software using the MPSoC asymmetric processing cores and Field Programmable Gate Array (FPGA) fabric. This is implemented in a manner that isolates the application processors running the flight software from interrupts associated with the input data. By leveraging real-time processors on the MPSoC, and a structure with the appropriate interfaces in the shared memory on the SBC, the flight software can use the full set of application processors. The available utilization for each processor in this set is also maximized for the SPLICE applications, providing a sufficiently deterministic execution environment without the cost and overhead of a real-time operating system.**

Keywords*— *Heterogeneous processing system, Multicore, Deterministic processing, Precision Landing and Hazard Avoidance

I. Introduction

NASA's Safe and Precise Landing – Integrated Capabilities Evolution (SPLICE) project develops sensor, algorithm, and compute technologies to enable Precision Landing and Hazard Avoidance (PL&HA) capabilities in support of future exploration missions [1]. The compute technology is in ongoing development as the SPLICE Descent and Landing Computer (DLC). Several versions of the DLC have been in development, including an early prototype [2], an Engineering Development Unit (EDU) flown on suborbital flight tests in cooperation with Blue Origin [3], and the current Engineering Test Unit (ETU) version targeted for lunar flight testing [4]. The DLC integrates the SPLICE sensors, algorithms, and computing resources to provide and integrated PL&HA payload for a host vehicle. The EDU DLC is shown in Fig. 1, with external interfaces.

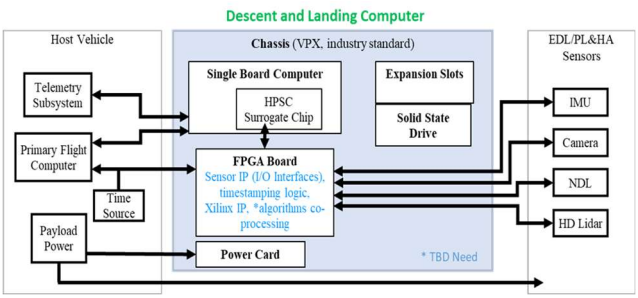

Fig. 1. EDU DLC with external interfaces.

The main components of the DLC shown in Fig. 1 are the Single Board Computer (SBC), where the flight algorithms are executed, and a Field-Programmable Gate Array (FPGA) board, which provides mission-specific sensor interfaces, and delivers the sensor data to the SBC for processing. The PL&HA sensor suite includes an Inertial Measurement Unit (IMU), a precision velocimeter called the Navigation Doppler LIDAR (NDL) [5], a camera for Terrain Relative Navigation (TRN), and a Hazard Detection LIDAR (HDL). These sensors provide data to the

SBC at different rates and packet sizes. Table 1 shows a representative example of these specifications.

TABLE I. PL&HA SENSOR DATA BANDWIDTH

Sensor	*Data Rate (Hz)*	*Packet Size (Bytes)*
IMU	400	48
NDL	20	280
Camera	10	1.6M
HDL	1	30M

The SBC and FPGA boards are NASA in-house designs that host Xilinx Multi-Processing System on a Chip (MPSoC) Ultrascale+ and Kintex Ultrascale FPGA chips respectively. The Xilinx MPSoC chip was selected due to the similarity of its architecture to NASA's planned architecture for the High-Performance Space Computer (HPSC) [6]. The HPSC had several years lead time at the time of this selection. The MPSoC has served as a surrogate to the HPSC, enabling years of valuable development and testing that will facilitate the transition to using an HPSC SBC in future systems.

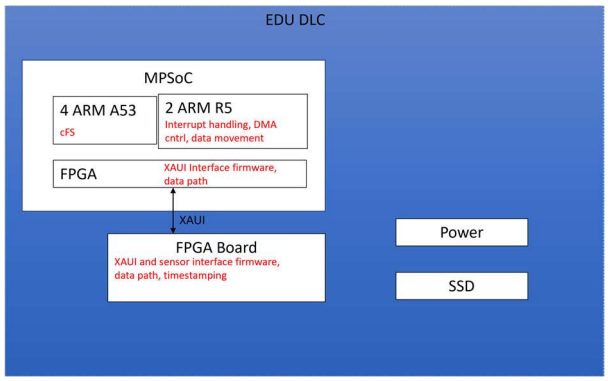

Fig. 2. MPSoC architecture and functional allocations.

The MPSoC architecture is shown in Fig. 2. It is a heterogeneous multicore processing system, with a cluster of 4 ARM A53 processors, 2 ARM R5 real-time processors, and FPGA fabric tightly integrated as a System on a Chip [7]. Not shown is a GPU also present on the chip, that was not used for our application since a GPU is not included in the planned HPSC. Shown in Fig. 2, the flight software runs on the A53 cluster, marked "cFS" for the Core Flight System framework used [8]. The Xilinx Petalinux embedded Linux operating system is used [9]. The R5 cluster is used for processing lower level, closer to the hardware functions such as interrupt handling, Direct Memory Access (DMA) control and data movement. The FPGA fabric implements the MPSoC side of the sensor data path that also includes the FPGA board, with a high-speed serial 10 Gigabit Attachment Unit Interface (XAUI) link between the boards. The FPGA board implements the other side of the sensor data path, including sensor specific interfaces and timestamping. The following sections describe how this heterogeneous architecture is used to implement the contribution of this work.

II. Related Work

The SPLICE algorithms are complex and must execute with low latency and in a deterministic manner to function real-time in a landing system. Key requirements of the data processing system are determinism (the system responds in a predictable and consistent manner), and resource efficiency (it is critical to reserve as much of the A53 processor time as possible for SPLICE algorithms). This can be accomplished by a system that does not interrupt the application processors with the input data. An option in multicore systems is to dedicate one or more application processors to handling interrupts from the input data. This approach reduces the set of available processors and processing capacity for the application. Expensive Real-Time Operating Systems (RTOS) can be incorporated to manage a system on a schedule that minimizes the impact of input data interruptions [10]. SPLICE is a small technology development program and did not have a budget for an RTOS. The R5 real-time processors available in the MPSoC provided an alternative approach for SPLICE. These processors are not suitable for the application processing, or even to host a traditional operating system. But they are suitable for operations closer to the hardware, as they are specifically designed to provide reliable, low latency, deterministic operation for these real-time functions.

A key distinction of the technology in this application is that the OpenAMP [11] template Xilinx recommends to setup communication between the real-time and application processors was not used. An early prototype of this system that used OpenAMP resulted in having an Interprocessor Interrupt (IPI), or software interrupt between the real-time and application processors [2]. This only achieved a reduction of interrupts to the application. This interruption was not only an undesired performance hit, but these interrupts, forcing the applications to process data when the data arrives in real-time, is unacceptable to the operation of the SPLICE algorithms. The SPLICE algorithms needed to get new input data on their own schedule, and not have the schedule driven by the data availability.

III. Datapath Design

Fig. 3 shows a block diagram of the SPLICE input data path. The data path spans the FPGA and MPSoC boards connected with a high-speed serial link. The FPGA board combines the individual sensor inputs into a single multiplexed data stream. The MPSoC board receives the data stream and delivers each sensor's data to the DRAM memory on the board from where the flight algorithms retrieve and process it.

On the FPGA board, IMU, NDL, Camera and HD Lidar IP facilitate the harvesting of sensor data from disparate sources with varied clock and packet rates. Time stamping of the

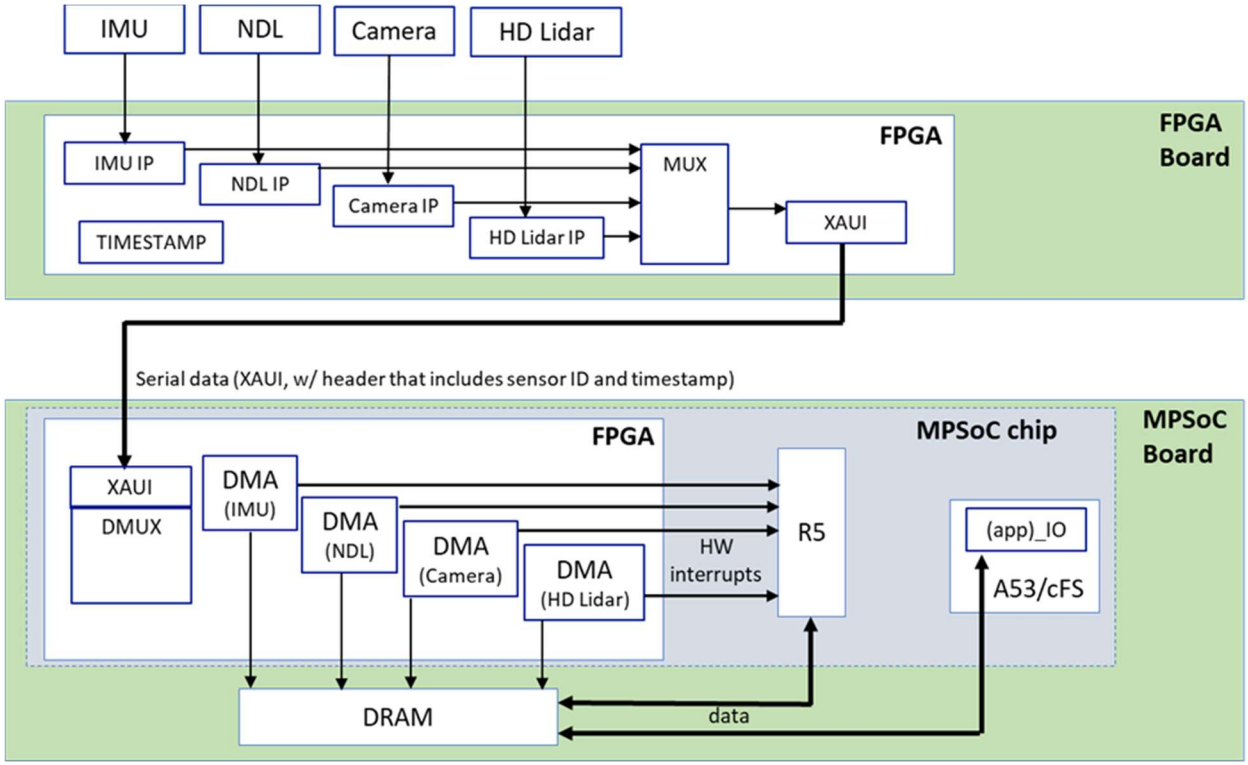

Fig. 3. SPLICE datapath.

received data packets and the reformatting of data into the Advanced eXtensible Interface (AXI) 4-Stream protocol is performed [12]. Asynchronous AXI4-Stream (AXIS) FIFO IP are then used to convert individual sensor channel pipelines into the XAUI clock domain before the channels are multiplexed onto a single AXIS bus. This combined bus passes data to the Kintex's outbound XAUI interface.

The combined sensor data stream input to the MPSoC board is demultiplexed to separate streams and handled by separate Direct Memory Access (DMA) Commercial Off The Shelf (COTS) Intellectual Property (IP) circuits. These DMA IP are configured to transfer incoming data into the DRAM, and to provide interrupts upon completing each transfer which are handled by embedded software running on the Real-Time (R5) processors. The DMA IP are configured differently for different data rates. For smaller packet size, higher-rate data, such as the IMU and NDL, a scatter/gather mode is used for DMA transfers. This mode sets up a hardware "loop" to manage the transfers with minimal interaction from the application running on the R5 processors [13]. Conversely, the large data transfer and lower-rate data, such as the camera and HDL, use a simple DMA transfer configuration which allows for a higher bandwidth data transfer.

Fig. 4 is an event diagram showing the sequence of actions performed by each data path system component, after the combined sensor input data stream is demultiplexed and the individual sensor data arrives at each respective DMA component on the MPSoC board. Starting in the upper left of the figure and proceeding clockwise, the dotted-line boxes represent a software program running on the real-time processors, the custom logic design in the MPSoC FPGA, the flight software application running on the application processors, and a ring buffer structure in the memory resident on the MPSoC board.

Referring the block designated R1 in Fig. 4, the real-time processor application begins by initializing the DMA components and configuring them to begin transfers (block R2). Data begins arriving at the DMA components for which events begin shown in blocks L1a and L1b in the figure. The a and b branches of the figure represent the two different operational configurations of the DMA components, Scatter/Gather and Simple DMA, respectively. In the case of Scatter/Gather, the DMA receives data (block L1a), and copies the data to descriptor buffers (special buffers allocated in the memory for this DMA configuration) which corresponds to block L2a in the figure. For simple DMA, the component must receive a signal to initiate the transfer (block L1b) upon arrival of the data (block L2b). Upon arrival of the data, the simple DMA component copies the data to the appropriate location in the memory, shown in block L3b. Both DMA configurations conclude this data transfer sequence by sending a hardware interrupt to the real-time processors (block L4).

Blocks R3a and R3b show the real-time processor application for the Scatter/Gather and Simple DMA configurations receiving the hardware interrupt from the custom logic. For

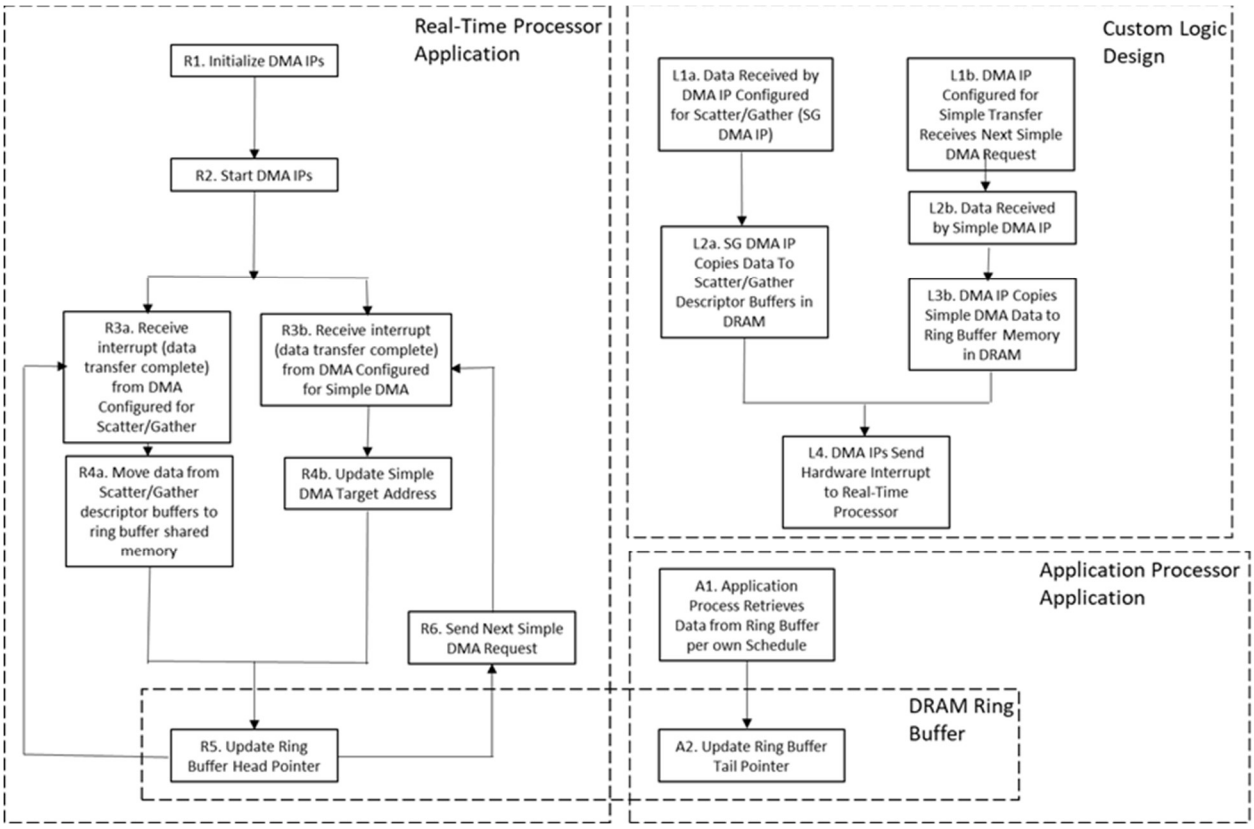

Fig. 4. Datapath event sequence.

Scatter/Gather, the application then transfers the data from the descriptor buffers in memory to the appropriate locations in the ring buffer (block R4a). For simple DMA, the next address is calculated for use in the next simple DMA transfer request (block R4b). To achieve the desired zero interrupts to the application processors, the OpenAMP IPI was abandoned, and a ring buffer structure found in the literature [14] was incorporated into the DRAM. This ring buffer incorporates two pointers to the memory, a head pointer and a tail pointer. The real-time processor updates the head pointer when data is written to the buffer from the hardware, and the application updates the tail pointer when data is read from the buffer for use by the application. A diagram of the ring buffer is shown in Fig. 5.

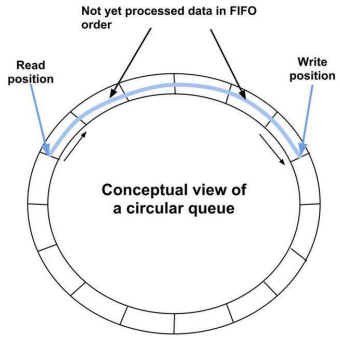

Fig. 5. Ring buffer queue.

Returning to Fig. 4, the real-time application updates the head pointer in the ring buffer structure, shown in block R5. For simple DMA, the application must send the next simple DMA transfer request, shown in block R6. This event concludes the activities for the data path design for a single transfer of the required data to memory for a user application.

A user application uses an internal schedule for reading the data from the memory and initiates a read per that schedule as shown in block A1 of Fig. 4. When the read is complete, the application updates the tail pointer to the ring buffer, shown in block A2. This completes a single transfer of data from the memory to user algorithms. These events of Fig. 4 repeat continuously for subsequent data arriving from the sensors. This implementation achieves the goal of providing full isolation of the application from interrupts associated with the input data.

IV. Discussion

The SPLICE data path design was used in the EDU DLC which flew on two suborbital flight tests on Blue Origin's New Shephard booster [3]. There was approximately one year between the two flights. On the first flight, all the DMAs were configured for scatter/gather operation. With this design, the largest bandwidth camera data (the HDL sensor was not integrated for these flights) could only be collected at a

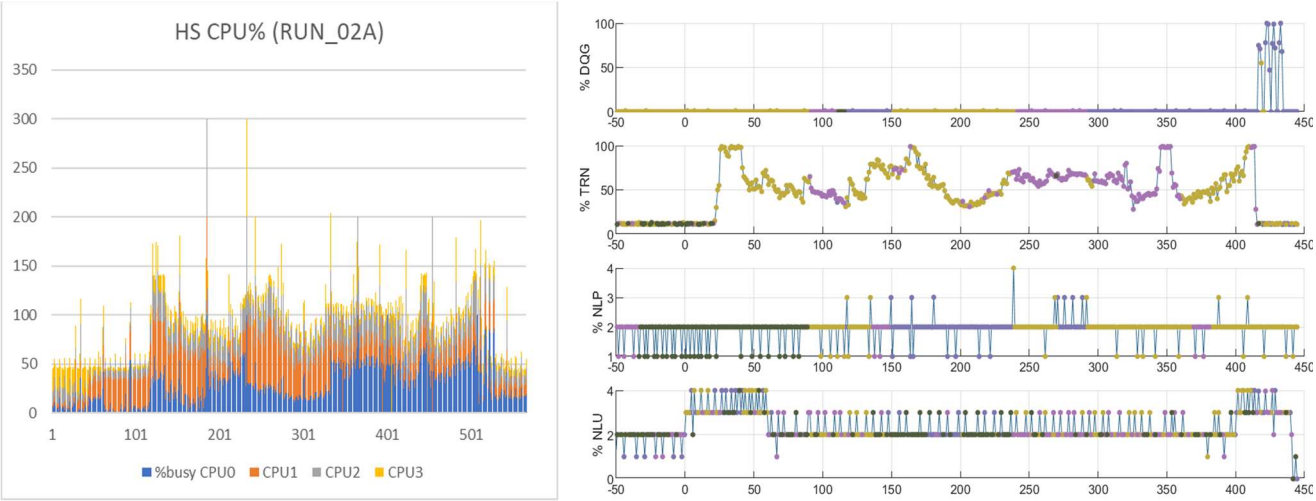

Fig. 6. EDU DLC processor utilization during simulated SPLICE flight.

maximum of 2 frames per second. Higher frame rates caused the datapath to stop operating after an amount of time that was not predictable. It was discovered that the R5 operation of moving the data from the scatter/gather descriptor buffers to the ring buffer in the shared memory was a bottleneck. After the camera data was changed to a simple DMA transfer, the project goal of 10 frames per second camera data was achieved. This final design for handling the multi-rate sensor data flew successfully on the second Blue Origin/SPLICE test.

The data path was tested for adverse effects on the deterministic operation of the flight software. Simulated sensor data was run through the design continuously for several days, without any data packet drops or corrupted data. In addition, a high-fidelity Hardware-In-The-Loop simulation was constructed and over a dozen simulated flights using trajectories from the two suborbital flights were performed. No non-deterministic behavior of the flight software that could be attributed to the data path was observed in either the actual or simulated flights. Performance metrics were collected both in flight and in the simulations, which included the utilization of the application processors [3]. Fig. 6 shows this utilization as a stacked bar graph for the four A53 processors in the MPSoC. As shown in the left graph, the overall utilization running all the SPLICE PL&HA algorithms in the cFS framework was almost never over 50%, and remained close to 25% for the majority of the flight. The right set of graphs show the utilization for each of the navigation algorithms tested. Comparing the two, the right graph shows that the algorithms account for approximately 15% of the overall utilization, and the left graph shows an average utilization of about 20%. The overhead of everything in the cFS framework (I/O, utilities, telemetry, housekeeping, scheduler, etc.) and Linux operating system, beyond the navigation algorithms, is about 5%. This result indicates that the design is preserving a high percentage of the processing capacity for the PL&HA application algorithms.

V. Conclusions and Future Work

Results of testing the SPLICE EDU data path indicate that the design supports deterministic flight software operation while preserving application computing capacity on the DLC platform. This is accomplished without the use of a real-time operating system. The data path is being used in the next version of the DLC that is being designed to support a robotic lunar test flight. Enhancements planned are additional channels for the Hazard Detection LIDAR sensor and a vehicle interface. Additional enhancements will be made to the command path of the interface, which was not discussed here.

References

[1] R. Sostaric, et al, "The SPLICE project: safe and precise landing Technology development and testing," Proceedings of the AIAA SciTech Forum (virtual), January 2021.

[2] D. Rutishauser, J. Prothro, R. Moore, H. Yim, "High performance computing for precision landing and hazard avoidance and co-design approach," IEEE Aerospace Conference, Big Sky MT, March 2019.

[3] D. Rutishauser, G. Mendeck, R. Ramadorai, J. Prothro, T. Fleming, and P. Fidelman, "NASA and Blue Origin's flight assessment of precision landing algorithms computing performance," Proceedings of the AIAA SciTech Forum, San Diego, January 2022.

[4] D. Rutishauser, H. Yim, "Safe and Precise Landing – Integrated Capabilities Evolution (SPLICE) Descent and Landing Computer (DLC)," IEEE Space Computing Conference (virtual), August 2021.

[5] A. Gragossian, D. Pierrottet, J. Estes, B. Barnes, F. Amzajerdian, G. Hines, "Navigation Doppler Lidar performance at high speed and long range," Proceedings of the AIAA Scitech Forum, January 2020.

[6] Richard Doyle, Raphael Some, Wesley Powell, Gabriel Mounce, Montgomery Goforth, Stephen Horan, and Michael Lowry, "High Performance Spaceflight Computing; Next-Generation Space Processor: A Joint Investment of NASA and AFRL," International Symposium on Artificial Intelligence, Robotics, and Automation in Space (iSAIRAS 2014), Montreal, Canada, June 2014.

[7] Xilinx MPSoC Web site: https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html Accessed: 2023-01-23

[8] The core Flight System (cFS) Web site: https://cfs.gsfc.nasa.gov/ Accessed: 2023-01-23

[9] Xilinx Petalinux Web site: https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html Accessed: 2023-01-23

[10] Windriver Web site: https://www.windriver.com/products/vxworks Accessed: 2023-01-23

[11] Notes on Asymmetric Multiprocessing – AMP | Kynetics – Technical Notes https://technotes.kynetics.com/2018/Notes_on_AMP/ Accessed: 2023-01-23

[12] Xilinx AXI4 Web site: https://www.xilinx.com/products/intellectual-property/axi.html Accessed: 2023-01-23

[13] Xilinx AXI DMA Produce Guide Web site: https://docs.xilinx.com/r/en-US/pg021_axi_dma/Direct-Register-Mode-Simple-DMA Accessed: 2023-01-23

[14] GitHub Website https://github.com/enfiskutensykkel/lamport Accessed: 2023-01-23